DYNAMIC SOFTWARE VERSION AUDITOR WHICH MONITORS A PROCESS TO PROVIDE A LIST OF OBJECTS THAT ARE ACCESSED

BACKGROUND OF THE INVENTION

For many years, there was little in the way of automated "configuration management" in relation to computer software development. As the early software "systems" were developed, documentation and control of the "current version" was most often accomplished as a de-facto manual configuration management. A system was built, tested, the components revised and the system rebuilt. Finally, the system was installed and any future changes were made by way of "patches" to the software system.

More recently, Computer-Aided Software Engineering (CASE) environments have become helpful for complex software projects, just as Computer-Aided Design (CAD) systems have become helpful for complex hardware projects. A few well known CASE systems is include: UNIX/PWB,designed to run on AT&Ts UNIX programming environment, includes the SCCS source code control system and the MAKE configuration tool; RCS, a source code control system that also runs on UNIX systems; CMS and MMS, the Digital Equipment Corp. VAX/VMS equivalent to SCCS and MAKE; ALS, the Ada Language System; and Cedar, running on the Xerox PARC Computer Science Laboratory system.

While prior art CASE systems as described above have offered an improvement in the ability to keep track of various configurations of software systems as they are built and modified, the have limitations. One drawback is the lack of "transparency" and "concurrency" to users in the area of configuration management. For instance, the Master program source codes (i.e., the basic or first versions of the programs) are typically maintained in a Master Table and changes by version are maintained in a Versions Table. The various possible versions of a program (or system comprising multiple programs) are not transparently available to users. Thus, to build a given configuration of a system, a System Builder cannot directly access the versions of the programs to be used in the system build. Rather, control must first be given to a Version Maker which then accesses the Master Table to obtain the master program source(s) and then gets and applies the appropriate version changes from the Version Table. The modified program reflecting a designated version is then stored in a holding area. Finally, control is transferred to a System Builder, which get its input from the holding area, from which it "builds" the desired system in a build area. Moreover, unless multiple holding areas and attendant complex procedures for their use are provided, there is a lack of concurrency in that only one version of the system can be built at any one time.

Another shortcoming of prior art CASE systems is the lack of capability to track and report progress on tasks or to monitor and notify of changes in areas critical to others.

SUMMARY OF THE INVENTION

It is the object of the present invention to provide a CASE system providing transparent access to multiple versions, the ability to build different configurations concurrently without interference, and additional monitoring and reporting capabilities not found in known CASE systems.

The present invention features a CASE version-control system that supports versioning of all file system objects: files, directories, and links. Any type of file can be versioned, including executables, bitmaps, and other non-text files. Versions of directories record how the organization of the source base evolves: renaming of source files, creation of new source files, and so on.

The present invention also features Rule-Based Version Selection in which users can create and use any number of views, each of which selects a particular configuration of source versions. Views are defined by configuration specs consisting of a few powerful, general rules. Thus, there is no need to specify hundreds or thousands of source versions individually. Views are dynamic, updated by reevaluating the rules that define it as needed. Newly-created versions can thus be incorporated into a view automatically and instantly.

The system of the present invention further provides Transparent Access to versions of objects. Each versioned object appears to be an ordinary file or directory. This transparency feature makes the system compatible with common operating systems such as the UNIX open-systems environment. Developers can continue to use their existing tools—shells, editors, compilers, debuggers, and so on—no modifications need be made to such tools.

The present invention also features configuration auditing which automatically produces configuration records, which provide complete "bill-of-materials" documentation of software builds. Each configuration record includes a listing of all source file versions that were used, versions of build tools, and all build options that were specified. Special commands compare configuration records, showing the differences between two builds of the same program. Other commands can place version labels on object versions listed in the configuration record. Whenever possible, the present invention shares the derived objects produced by builds among users. This saves both time and disk storage. In a build that involves execution of multiple makefile build scripts, the scripts can execute in parallel, either on a single host or on a group of hosts in the local area network. A view's configuration spec can be defined in terms of the configuration records produced by previous builds. The exact source base for an individual program or an entire release can be recreated instantly in a new view, thus guaranteeing rebuildability of software systems.

The system minimizes data duplication, both for source files (elements) and for build targets (derived objects). A source file version is copied only when a developer wishes to modify it. If a build script would create a redundant copy of an object module or executable, the system automatically creates a link to an existing instance. Versions of text files are stored efficiently as deltas, much like SCCS or RCS versions. Versions of non-text files are also stored efficiently, using data compression.

In general, in one aspect, this invention features a data processing system and method for controlling versions of data, including a processor for executing instructions and for retrieving data objects from and storing objects to a storage device, a storage device for storing versions of objects, and an object version selector for providing the processor with access only to specific versions of target data objects as determined by a set of selection rules. The selection rules are evaluated for an object when that object is accessed by the processor. Preferably, the version selector includes a means for viewing the selected versions of the target objects as a transparent file system having directories, files, and links.

In preferred embodiments, each data object has a pathname for accessing the object from the data storage device,